Python and Scipy

Computer techniques for modellers

David Ham

David.Ham@imperial.ac.uk

Imperial College London

Contents

- What is python?
- Basic syntax and data types
- Modules
- Python for Nump(t)ys
- Pylab

Python is an **interpreted**, **object-oriented** scripting language.

It features powerful intrinsic data types and a massive library of modules providing ready made solutions to a huge range of programming tasks.

Python's syntax is very simple and its powerful error handling system as well as the interpreted nature of the language make it easy to debug.

Guido van Rossum, Python Benevolent Dictator For Life

Interfaces for Python

```
dham@attila python > ipython
Python 2.4.4 (#2, Apr 25 2007, 22:41:41)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.0 -- An enhanced Interactive Python.
?         -> Introduction to IPython's features.
%magic    -> Information about IPython's 'magic' % functions.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]: 1/2
Out[1]: 0

In [2]: a=1./2

In [3]: print a==0
False

```

Python scripts

A Python script is just a text file:

```
#!/usr/bin/python

planet="world"

print "Hello "+planet
```

Just like with all executables, under Unix you have to set the executable permission bit:

```
dham@attila python > chmod a+x hello
dham@attila python > ./hello
Hello world
dham@attila python >
```

Scalar data types

Python has all the usual numeric and logical types:

```
In [1]: type(1)
Out[1]: <type 'int'>

In [2]: type(1.0)
Out[2]: <type 'float'>

In [3]: type(1.0+3.0j)
Out[3]: <type 'complex'>

In [4]: type(True)
Out[4]: <type 'bool'>
```

This is a good time to introduce the wonders of the Python online library reference.

Dynamic typing

Like most interpreted languages, you don't declare the type of names in Python. Every variable name has the type it is currently assigned to:

```
In [1]: a=1.0; type(a)
Out[1]: <type 'float'>

In [2]: a=1; type(a)
Out[2]: <type 'int'>
```

In many circumstances, type is not important. As long as the operations requested are defined for the current type of a variable, everything just works.

Sequence types

Python has several types of sequence type. Two of the most important are tuples and lists.

Lists, written [item1, item2, item3] are *mutable*. This means that you can change an existing list object:

```
In [1]: a=[1,2]

In [2]: a[1]='f'

In [3]: a
Out[3]: [1, 'f']
```

Note that sequences in Python are indexed from a base of zero.

Immutable sequence types

Tuples are immutable. Once created you can't change them:

```
In [4]: b=(1,2)

In [5]: b[0]=3
```

```
exceptions.TypeError
Traceback (most recent call last)

/home/dham/<ipython console>

TypeError: object does not support item assignment
```

The difference between immutable and mutable types is an important concept in Python. Mutable types are clearly very flexible but in some circumstances, only immutable types are allowed.

Assignment on mutable types

Assignment of mutable types does not make a copy!

```
In [33]: a=[1,2]

In [34]: b=a

In [35]: b[1]=3

In [36]: a
Out[36]: [1, 3]
```

Strings

Strings are immutable sequences of characters:

```
"this is a literal string"
'so is this'
'''This string can contain literal
line breaks.'''
'special characters can be included using backslashes\n'
r"the raw string syntax (with a leading r) is used when you want all the
characters to have their literal meanings"
```

Almost any type can be converted to a string using the backquote ‘ operator:

```
In [11]: a=1

In [12]: print "Python is number "+‘a‘
Python is number 1
```

Dictionaries

A dictionary is an array indexed by keys instead of numbers. A key can be any immutable data type. For example, you can actually use a (Python) dictionary as a (language) dictionary:

```
In [13]: dict={}

In [14]: dict['banana']="A bendy fruit"

In [15]: dict[42]="The answer"

In [16]: dict.keys()
Out[16]: [42, 'banana']

In [17]: dict[42]
Out[17]: 'The answer'
```

Sets

Mathematicians will be delighted to know that since version 2.4, python has sets:

```
In [25]: a=set(("apple", "pear"))

In [26]: b=set(("apple", "orange"))

In [27]: a|b
Out[27]: set(['orange', 'pear', 'apple'])

In [28]: a&b
Out[28]: set(['apple'])
```

Modules

All those nifty Python Modules in the Standard Library are accessed using the “import” command. For example:

```
#!/usr/bin/python
import sys # Whole load of system commands.

# This is the correct way for a Python program to die:
sys.exit(1)
```

You can also import the whole module into the current namespace:

```
#!/usr/bin/python
from numpy import * # Just a little taster of numpy.
```

Importing * from a module can really muck up you namespace. Do it with caution!

Indentation and blocks

Python uses indentation to indicate code block bodies. For example, for loops iterate over a sequence:

```
for key in dict.keys():
     print dict[key]
# Note there is no end of loop indicator.
```

This applies to nested loops:

```
for p1 in points:
    for p2 in points:
        if p1<p2:
            # Insert each set only once.
            edgeset.add((p1,p2))
```

Our own little script

Suppose we have a file containing the connectivity of a mesh. It looks like this:

```
19  4  0
    1       16     11     14     15
    2        9     10      1     11
    3        5      9      1     11
    4       12      6     13     15
...
   16       11     16     14     10
   17       16     15     14      3
   18       11     16     10      8
   19       11     15     13     14
# Generated by tetgen -e cube.poly
```

The first line is a header. The other lines list the nodes in each element. We wish to count the number of edges in this mesh. Of course we should also ignore the comment line.

Numpy

Numpy is the core of Scipy. It adds matrix and array operations to Python. There are two key data types:

matrix always 2D, * is matrix multiply.

array arbitrary dimensions, * is element-wise (Matlab .*)

At the moment, there is no native syntax for writing arrays and matrices so you build them out of lists of lists:

```
In [43]: from numpy import *

In [44]: a=matrix([[1,2,1],[3,4,3]])

In [46]: a
Out[46]:
matrix([[1, 2, 1],
        [3, 4, 3]])
```

Matrix and array methods

```
In [15]: a=matrix([1,2,3]).reshape(3,1)

In [16]: a.T*a
Out[16]: matrix([[14]])

In [21]: b=matrix([[2,1],[1,2]])

In [22]: linalg.inv(b)
Out[22]:
matrix([[ 0.66666667, -0.33333333],
        [-0.33333333,  0.66666667]])

In [23]: linalg.eig(b)
Out[23]:
(array([ 3.,  1.]),
 matrix([[ 0.70710678, -0.70710678],
         [ 0.70710678,  0.70710678]]))
```

Pylab and matplotlib

The matplotlib library provides the Pylab package which is a Matlab-like plotting interface. It works much better with python shell support. Eg:

```
david@ese-dhamlap:~$ ipython -pylab

In [1]: from numpy import *

In [2]: a=2*math.pi*matrix(range(11)).T/10

In [3]: plot(a, sin(a), '-')
```